Verilog Test Bench Design Code

Anand Rajan

Pennsylvania State University, University Park

Section 002R

CMPEN 331

Verilog Design Code

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company: Pennsylvania State University, University Park

// Engineer: Anand Rajan

//

// Create Date: 04/04/2021 08:09:09 PM

// Design Name:

// Module Name: top

// Project Name: Lab 4

// Target Devices: XC7Z010-CLG400-1

// Tool Versions:

// Description: The project aims to develop a basic CPU through pipelining.

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module top(input clk, output mwreg\_sig, output mm2reg\_sig, output mwmem\_sig, output [4:0] mmuxout\_sig,

output[31:0] maluout\_sig, output [31:0] dm\_sig, output ealuimm\_sig, output [4:0] mux\_sig);

// Block 1: IF stage

wire[7:0] adder\_to\_pc; // Output from adder

wire[7:0] pc\_out; // Output from PC register

wire[31:0] im\_to\_ifid; // Output from instruction memory

// Block 2: ID stage

wire[31:0] ifid\_out;

// Outputs from the Control Unit

wire wreg\_to\_idexe;

wire m2reg\_to\_idexe;

wire wmem\_to\_idexe;

wire[3:0] aluc\_to\_idexe;

wire aluimm\_to\_idexe;

wire RegDst;

// Output from Mux

wire [4:0]mux\_to\_idexe;

// Output from Register File

wire [5:0]qa\_to\_idexe;

wire [5:0]qb\_to\_idexe;

// Output from Sign Extender

wire[31:0] eimm\_to\_idexe;

// Block 3: EXE stage (next lab)

wire ewreg;

wire em2reg;

wire ewmem;

wire[3:0] ealuc;

wire ealuimm;

wire [4:0] emuxout;

wire [5:0] eqa;

wire [5:0] eqb;

wire [31:0] eeimm;

wire [31:0] mux\_to\_alu;

wire [31:0] aluout;

// Block 4: MEM stage

wire mwreg;

wire mm2reg;

wire mwmem;

wire [4:0] mmuxout;

wire [31:0] maluout;

wire [5:0] mqb;

wire [31:0] dm\_to\_memwb;

// Block 5: WB stage

wire wwreg;

wire wm2reg;

wire [4:0] wmuxout;

wire [31:0] waluout;

wire [31:0] wdo;

// Module Instantiations

PC pc(clk, adder\_to\_pc, pc\_out);

Adder adder(pc\_out, adder\_to\_pc);

IM im(pc\_out, im\_to\_ifid);

IFID ifid(clk, im\_to\_ifid, ifid\_out);

CU cu(ifid\_out[31:26], ifid\_out[5:0], wreg\_to\_idexe, m2reg\_to\_idexe, wmem\_to\_idexe, aluc\_to\_idexe, aluimm\_to\_idexe, RegDst);

Mux mux(ifid\_out[15:11], ifid\_out[20:16], RegDst, mux\_to\_idexe);

Regfile regfile(ifid\_out[25:21], ifid\_out[20:16], qa\_to\_idexe, qb\_to\_idexe);

Signext ext(ifid\_out[15:0], eimm\_to\_idexe);

IDEXE idexe(clk, wreg\_to\_idexe, m2reg\_to\_idexe, wmem\_to\_idexe, aluc\_to\_idexe,

aluimm\_to\_idexe, mux\_to\_idexe, qa\_to\_idexe, qb\_to\_idexe, eimm\_to\_idexe,

ewreg, em2reg, ewmem, ealuc, ealuimm, emuxout, eqa, eqb, eeimm);

Mux2 mux2(eqb, eeimm, ealuimm, mux\_to\_alu);

ALU alu(eqa, mux\_to\_alu, ealuc, aluout);

EXEMEM exemem(clk, ewreg, em2reg, ewmem, emuxout, aluout, eqb,

mwreg, mm2reg, mwmem, mmuxout, maluout, mqb);

DataMem dm(maluout, mqb, mwmem, dm\_to\_memwb);

MEMWB memwb(clk, mwreg, mm2reg, mmuxout, maluout, dm\_to\_memwb,

wwreg, wm2reg, wmuxout, waluout, wdo);

// Since the instructions ask for signals written INTO MEMWB and FROM EXMEM:

assign mwreg\_sig = mwreg;

assign mm2reg\_sig = mm2reg;

assign mwmem\_sig = mwmem;

assign mmuxout\_sig = mmuxout;

assign maluout\_sig = maluout;

assign dm\_sig = dm\_to\_memwb;

// for purpose of completion

assign ealuimm\_sig = ealuimm;

assign mux\_sig = mux\_to\_alu;

endmodule

module PC(input clk, input[7:0] a, output reg[7:0] q);

// Wire to set the value of PC to its next value; PC is a register effectively

always @(posedge clk)

begin

q <= a;

end

initial begin

q = 8'd100;

end

endmodule

module Adder(input [7:0]a, output reg[7:0] q);

// This module is not clock-dependent. It adds the input from PC register to the constant 4 and returns it.

// PC register should take this value to set as new value.

wire[7:0] to\_add = 8'd4;

always @(\*)

begin

q <= (a + to\_add);

end

endmodule

module IM(input[7:0] addr, output reg [31:0] do);

reg [31:0] IM [0:511]; // Array of registers

integer a;

always @(\*)

begin

a = addr; // integer-cast

do = IM[a];

end

initial // Hardcoded based on the examples provided to be done

begin

IM[100] = 32'b10001100001000100000000000000000;

IM[104] = 32'b10001100001000110000000000000100;

IM[108] = 32'b10001100001001000000000000001000;

IM[112] = 32'b10001100001001010000000000001100;

end

endmodule

module IFID(input clk, input [31:0]a, output reg[31:0] q);

// This is also clock-dependent. Outputs whatever it's inputted at pos clock edge.

always @(posedge clk)

begin

q <= a;

end

endmodule

module CU(input [5:0] op, input [5:0] func,

output reg wreg, output reg m2reg, output reg wmem, output reg[3:0] aluc, output reg aluimm, output reg regrt);

always @(\*) begin

case(op) // Determination of these values depends on the opcode, and func for R-format

6'b000000: // R-format instruction

begin

wreg = 1'b1;

m2reg = 1'b0;

wmem = 1'b0;

regrt = 1'b1;

aluimm = 1'b0;

case(func) // Using truth table in Zybooks

6'b100000: // add

aluc = 4'b0010;

6'b100010: // subtract

aluc = 4'b0110;

6'b100100: // AND

aluc = 4'b0000;

6'b100101: // OR

aluc = 4'b0001;

6'b100110: // XOR

aluc = 4'b0010;

6'b000000: // shift left

aluc = 4'b0010;

6'b000010: // logical shift right

aluc = 4'b0110;

// Not in truth table

// 6'b000011: // arithmetic shift right

// aluc = 4'b0010;

// 6'b001000: // register jump

// aluc = 4'b0010;

endcase

end

// Commented cases remain to be done in second part of lab - only necessary cases done

// 6'b001000: // addi

// 6'b001100: // andi

// 6'b001101: // ori

// 6'b001110: // xori

6'b100011: // lw

begin

wreg = 1'b1;

m2reg = 1'b1;

wmem = 1'b0;

regrt = 1'b0;

aluimm = 1'b1;

aluc = 4'b0010;

end

6'b101011: // sw

begin

wreg = 1'b0;

m2reg = 1'bX;

wmem = 1'b1;

regrt = 1'bX;

aluimm = 1'b1;

aluc = 4'b0010;

end

// 6'b000100: // beq

// 6'b000101: // bne

// 6'b001111: // lui

// 6'b000010: // j

// 6'b000011: // jal

endcase

end

endmodule

module Mux(input [4:0]rd, input [4:0]rt, input regrt, output reg [4:0]muxout);

always @(\*) begin

if (regrt) // select rd if RegDst is 1

muxout <= rd;

else // select rt if RegDst is 0

muxout <= rt;

end

endmodule

module Regfile(input [4:0]rs, input [4:0]rt, output reg [5:0]qa, output reg [5:0]qb);

reg [31:0] regfile [0:31];

integer a;

integer b;

always @(\*) begin

// qa outputs for rs and qb outputs for rt

a = rs; // this is to integer-cast the binary input

b = rt;

qa <= regfile[a];

qb <= regfile[b];

end

// asked to initialize all values in the regfile to 0

integer i;

initial begin

for (i=0; i<32; i=i+1) begin

regfile[i] = 0;

end

end

endmodule

module Signext(input [15:0]imm, output reg [31:0]eimm);

// converts 16 bit imm to 32 bits

always @(\*) begin

eimm[31:0] <= { {16{imm[15]}}, imm[15:0] };

end

endmodule

module IDEXE(input clk, input wreg, input m2reg, input wmem, input [3:0]aluc, input aluimm, input [4:0]muxout, input [5:0]qa, input [5:0]qb, input [31:0]eimm,

output reg ewreg, output reg em2reg, output reg ewmem, output reg [3:0]ealuc, output reg ealuimm, output reg [4:0]emuxout,

output reg [5:0]eqa, output reg [5:0]eqb, output reg[31:0] eeimm);

// Nothing but a clock-dependent output-input setter

always @(posedge clk)

begin

ewreg <= wreg;

em2reg <= m2reg;

ewmem <= wmem;

ealuc <= aluc;

ealuimm <= aluimm;

emuxout <= muxout;

eqa <= qa;

eqb <= qb;

eeimm <= eimm;

end

endmodule

module Mux2(input [5:0] a, input [31:0] b, input sel, output reg [31:0] q);

// Not dependent on clock; chooses 0 or 1 based on ealuimm

always @(\*) begin

if (sel) // select eeimm if RegDst is 1

q <= b;

else // select eqb if RegDst is 0

q <= a;

end

endmodule

module ALU(input [5:0] a, input [31:0] b, input [3:0] aluc, output reg [31:0] r);

// Not dependent on clock; performs different ops based on aluc

always @(\*) begin

case(aluc) // Determination of these values depends on the aluc

4'b0000: // AND

begin

r <= a && b;

end

4'b0001: // OR

begin

r <= a || b;

end

4'b0010: // add

begin

r <= a + b;

end

4'b0010: // subtract

begin

r <= a - b;

end

4'b0010: // set on less than

begin

if (a < b) begin

r <= 32'b1;

end else begin

r <= 32'b0;

end

end

4'b0010: // NOR

begin

r <= ~(a || b);

end

endcase

end

endmodule

module EXEMEM(input clk, input ewreg, input em2reg, input ewmem, input [4:0]emuxout, input [31:0] aluout, input [5:0]eqb,

output reg mwreg, output reg mm2reg, output reg mwmem, output reg [4:0]mmuxout, output reg [31:0] maluout, output reg [5:0] mqb);

// Nothing but a clock-dependent output-input setter

always @(posedge clk)

begin

mwreg <= ewreg;

mm2reg <= em2reg;

mwmem <= ewmem;

mmuxout <= emuxout;

maluout <= aluout;

mqb <= eqb;

end

endmodule

module DataMem(input [31:0] a, input [5:0] di, input we, output reg [31:0] do);

// Not dependent on clock; array

reg [31:0] dmem [0:31];

integer x;

integer y;

always @(\*) begin

if (!we) // if we is 0, read

// do outputs value at address a

x = a; // this is to integer-cast the binary input

//b = rt;

do <= dmem[x];

// if we is 1, write, but this lab doesn't cover that

// input di seems relatively useless - function not specified in TB, lecture, or hints video

end

// asked to initialize all values in the dmem to those provided

integer i;

initial begin

dmem[0] = 'hA00000AA;

dmem[1] = 'h10000011;

dmem[2] = 'h20000022;

dmem[3] = 'h30000033;

dmem[4] = 'h40000044;

dmem[5] = 'h50000055;

dmem[6] = 'h60000066;

dmem[7] = 'h70000077;

dmem[8] = 'h80000088;

dmem[9] = 'h90000099;

end

endmodule

module MEMWB(input clk, input mwreg, input mm2reg, input [4:0] mmuxout, input [31:0] maluout, input [31:0] do,

output reg wwreg, output reg wm2reg, output reg [4:0] wmuxout, output reg [31:0] waluout, output reg [31:0] wdo);

// Nothing but a clock-dependent output-input setter

always @(posedge clk)

begin

wwreg <= mwreg;

wm2reg <= mm2reg;

wmuxout <= mmuxout;

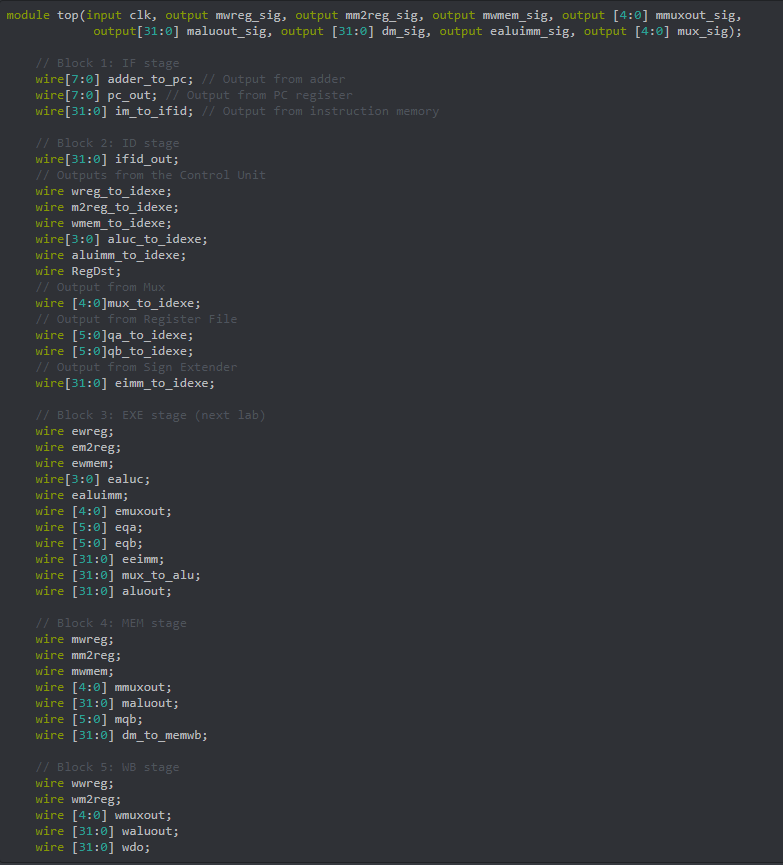
waluout <= maluout;

wdo <= do;

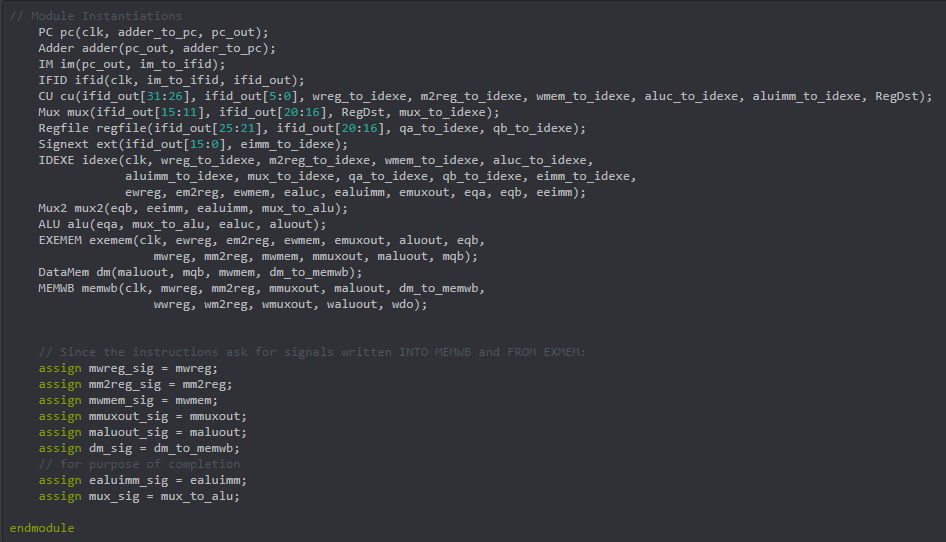
end

endmodule

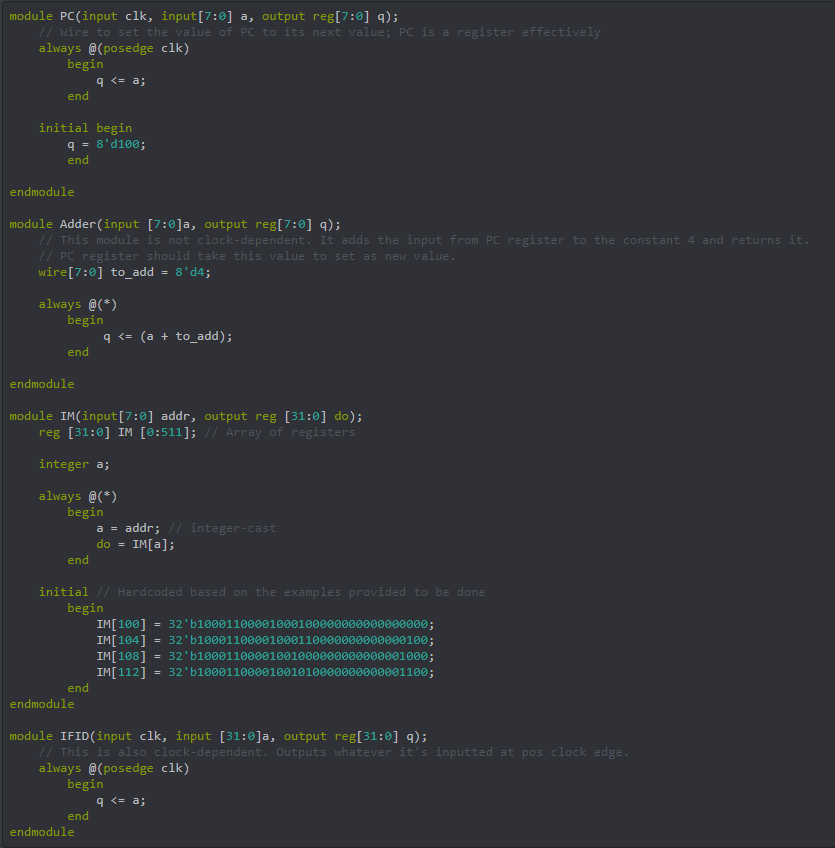
Note that the device used, as listed in the prefacing comments, is XC7Z010-1CLG400-1, as prescribed by the laboratory instructions. For readability purposes, screenshots of the same code are provided with keyword color highlighting as well as indentation displayed below (with the omission of the prefacing comments). The provision of the testbench code as well as screenshots of the same will follow this section.



*Image 1. Screenshot containing first section of Verilog design code – included for display, readability, and aesthetic purposes.*



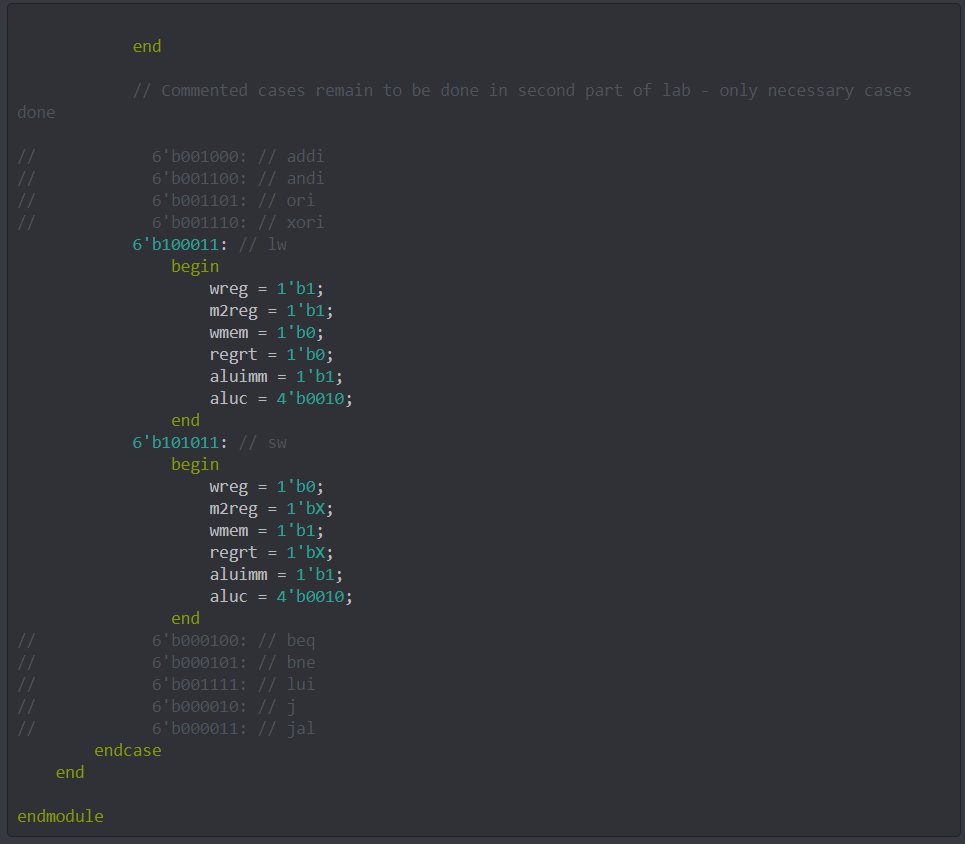
*Image 2. Screenshot containing second section of Verilog design code – included for display, readability, and aesthetic purposes.*



*Image 3. Screenshot containing third section of Verilog design code – included for display, readability, and aesthetic purposes.*



*Image 4. Screenshot containing fourth section of Verilog design code – included for display, readability, and aesthetic purposes.*



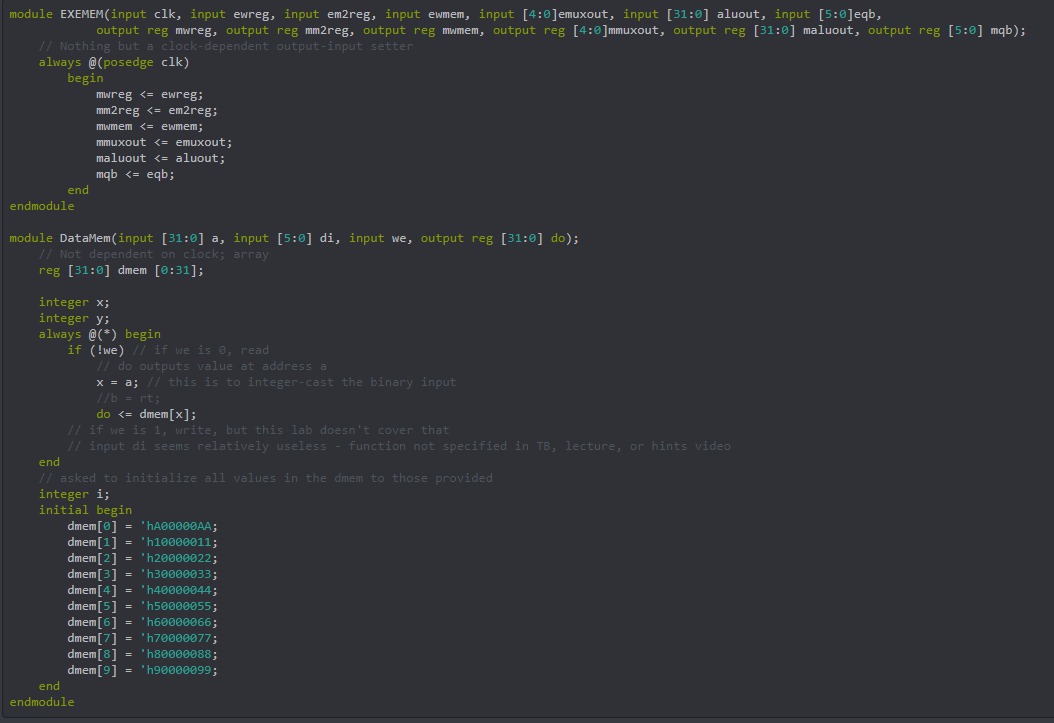
*Image 5. Screenshot containing fifth section of Verilog design code – included for display, readability, and aesthetic purposes.*



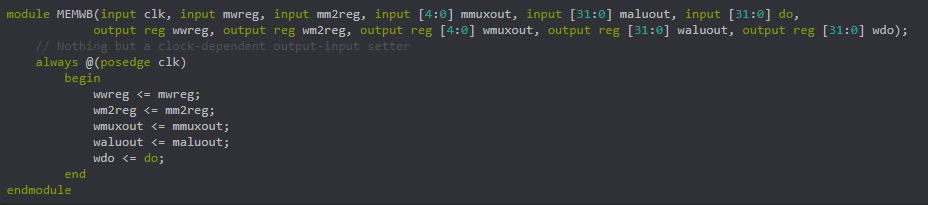
*Image 6. Screenshot containing sixth section of Verilog design code – included for display, readability, and aesthetic purposes.*



*Image 7. Screenshot containing seventh section of Verilog design code – included for display, readability, and aesthetic purposes.*



*Image 8. Screenshot containing eighth section of Verilog design code – included for display, readability, and aesthetic purposes.*



*Image 9. Screenshot containing ninth and final section of Verilog design code – included for display, readability, and aesthetic purposes.*

Verilog Test Bench Design Code

//////////////////////////////////////////////////////////////////////////////////

// Company: Pennsylvania State University, University Park

// Engineer: Anand Rajan

//

// Create Date: 04/04/2021 08:10:14 PM

// Design Name: Pipelining CPU

// Module Name: top

// Project Name: Lab 4

// Target Devices: XC7Z010-CLG400-1

// Tool Versions:

// Description: The project aims to develop a basic CPU through pipelining.

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

`timescale 1ns/1ps

module testbench();

reg clk\_tb = 0;

wire mwreg\_sig;

wire mm2reg\_sig;

wire mwmem\_sig;

wire[4:0] mmuxout\_sig;

wire [31:0] maluout\_sig;

wire[31:0] dm\_sig;

wire ealuimm\_sig;

wire [4:0] mux\_sig;

top dut(clk\_tb, mwreg\_sig, mm2reg\_sig, mwmem\_sig, mmuxout\_sig, maluout\_sig, dm\_sig, ealuimm\_sig, mux\_sig); // Initializing an instance of top

always begin

#5;

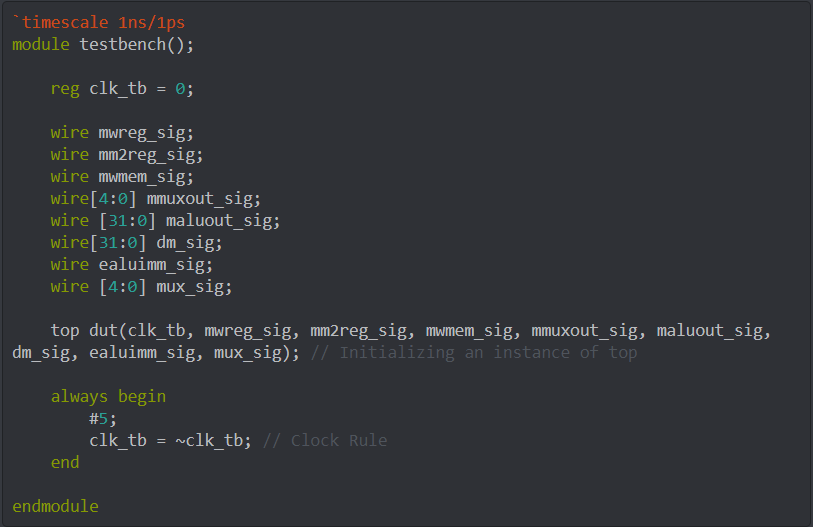
clk\_tb = ~clk\_tb; // Clock Rule

end

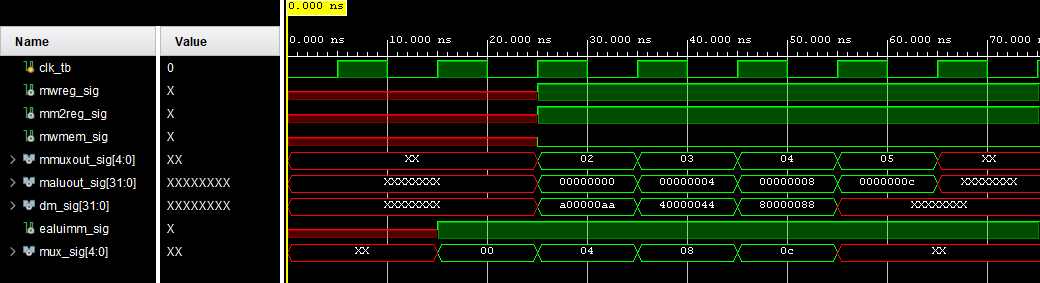
endmodule

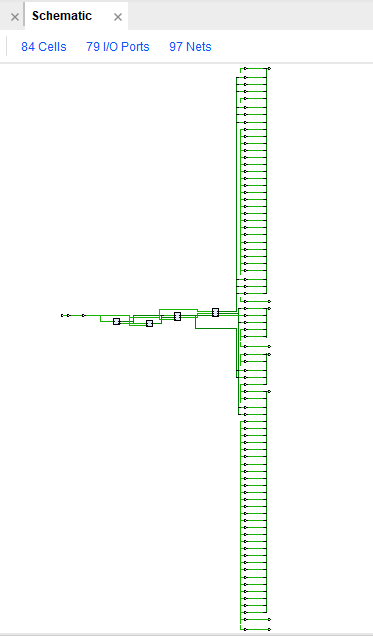
Note that this function really only requires the controlling of the clock through the testbench. These outputs are displayed as outputs for simplicity but the $display keyword could also have been used in place of this. Once again, for readability purposes, screenshots of the same code is provided with keyword color highlighting as well as indentation displayed below (with the omission of the prefacing comments).

The corresponding waveforms for the simulation of this design and testbench design code is also provided below the code screenshot. It should be stressed that the signals become X once more after 65 ns due to the lack of instructions past the PC’s of 100, 104, 108, and 112. Since most of these blocks perform calculations or selections, there is no selection to be made if there is no instruction derived from the IM. As such, they are X’s. The signals that come as outputs from the Control Unit extend forever PURELY BECAUSE OF THE CASE STATEMENT. No default statement was introduced since no such case was presented to account for this, and as such, with the lack of instructions, the control unit continues to output the same values as before infinitely. **THIS IS NOT A BUG. THIS CAN BE CHANGED WITH MORE INSTRUCTIONS OR SPECIFICATION OF A DEFAULT CASE.** All values for each signal are found on the signal and not on the left-hand sidebar. Refer to the signals themselves (those without values are logic signals).

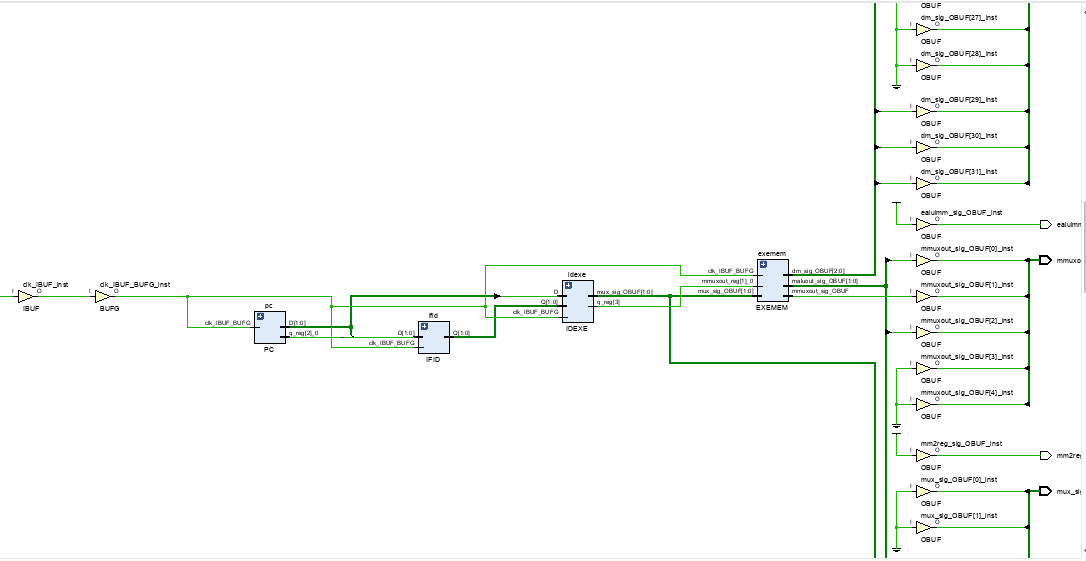


*Image 10. Screenshot of Verilog Test Bench Design Code provided for readability and aesthetic purposes*

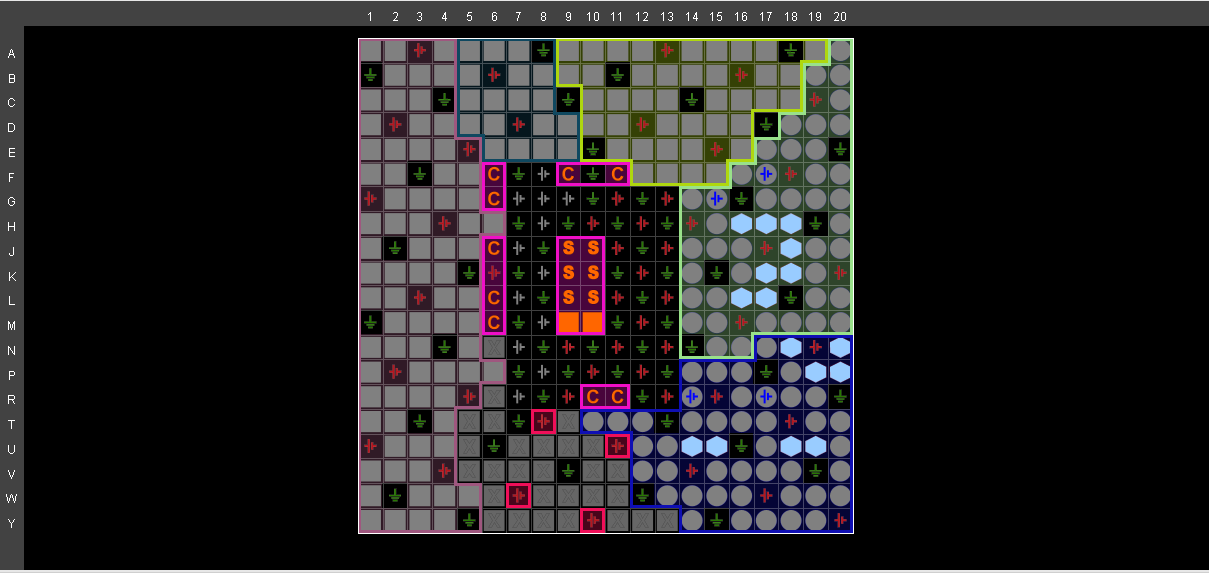
  
*Image 11. Screenshot of Final Waveform*



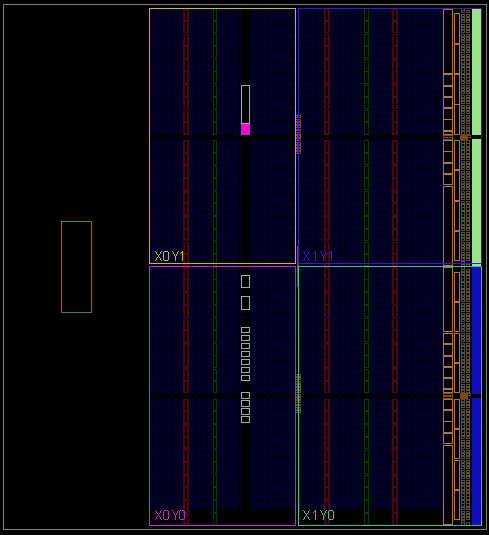
*Image 12. Design Schematics from the Synthesis of the Project (****SYNTHESIZED DESIGN –*** *xc7z010clg400-1)*



*Image 13. A close-up of what zooming into the center yields. Everything could not be zoomed in on for space’s sake.*



*Image 14. Snapshot of the I/O Planning*



*Image 15. Snapshot of the Floor Planning*

Images 12 through 16 provide the design schematic of the synthesized design and snapshots of the I/O and Floor planning of the same. It was difficult to discern which was which due to GUI errors, so the same choices of images were taken as the first lab. Note that the schematic provided could not be zoomed or resized in any way to make it clearer or larger. The information of the schematic is provided just above the design, in the same bar where the number of cells and such are mentioned. Specifics can be conducted within the project itself. The vertical layout was indeterminable though, and as such, cannot be zoomed in on without cutting a large portion out (or without having to post 20 or so snapshots).

With this, the requirements specified by the lab’s instructions are concluded. To recap, the specified device was used in this lab, which was done in Vivado. The Verilog design code as well as test bench code were both provided (albeit not with the 2x line spacing condition in both cases, but with 1x in the larger one in order to fit the code into as compact a space as possible – which still ended up being large). The code accounted for the blocks 1 and 2 of the basic CPU requested. Snapshots of the same were also provided in order to supplement readability and aesthetics such as indentations and comments. The waveform resulting from the verification of my design using the simulation software were also provided, with all the specified signals shown. The waveforms cover a timing period from 0 till 65 ns, since this is all the time necessary for 5-6 clock cycles (by which time the instructions will have come to pass). Additionally, the design was synthesized and the design schematics of this as well as the I/O and Floor planning designs were provided in snapshots (images 12 through 16). The cover page requirements were met to the best of my ability, and this project will be uploaded as a Word file (with no use of LaTex).